CONVERSE :
An Interoperable Framework for Parallel Programming

Laxmikant Kale, Milind Bhandarkar,

Narain Jagathesan, Sanjeev Krishnan, Joshua Yelon

Parallel Programming Laboratory

Department of Computer Science

University of Illinois, Urbana http://charm.cs.uiuc.edu

What is CONVERSE?

CONVERSE is an *interoperable parallel runtime system* that is designed to support execution of programs with *modules written in different parallel languages*.

Motivation

- Several different parallel programming paradigms exist
  - SPMD
  - Concurrent Objects: (Charm/Charm++, PC++/CC++, EC++, ABC++, ..)
  - Threads and Shared Memory
  - Data parallel
  - Distributed Shared memory
  - Others
- Each is well suited for a different type of application
- Each provides different run-time primitives
  - SPMD : tag based messages
  - Concurrent Objects : asynchronous remote method invocation

Motivation Cont.

- No single “best” paradigm exists
- Different paradigms have different advantages

One should be able to :

- express different modules in a large application in different paradigms
- combine pre-written modules from different paradigms

Interoperability helps Modularity and Reuse

CONVERSE Design

Desirables :

- Completeness : should support most paradigms
- Need based cost : each paradigm should incur cost only for its own features
- Efficiency : should be comparable to native (non-interoperable) implementation

Component based framework :

- Each component is specified by a concrete interface
- Each component can be implemented in different ways

Control Paradigms

(Examples: SPMD, Concurrent Objects, Threads, Data Parallel, Functional, ..)

Two characteristics :

- Concurrency within a process (processor)
  - No concurrency : SPMD
  - Concurrency via objects / threads
- Control regime:
  - Explicit, static : SPMD
  - Implicit, adaptive : Concurrent Objects, threads

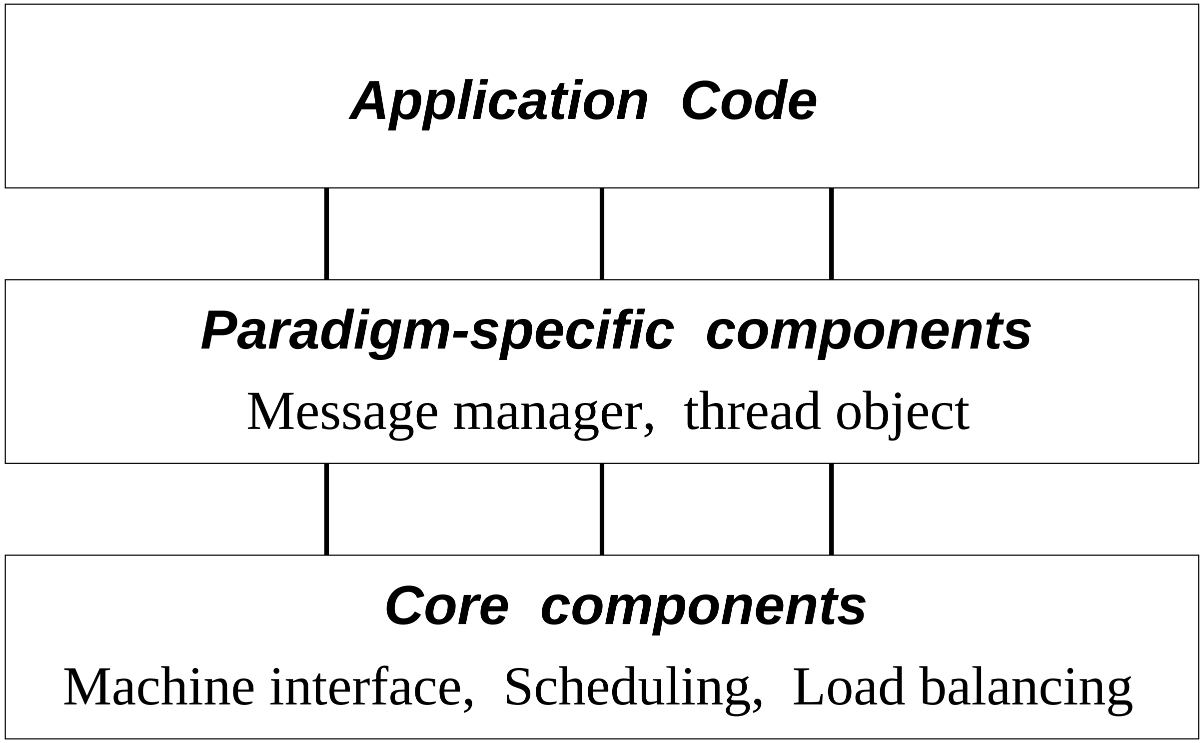
CONVERSE Scheduler

For implicit control regimes

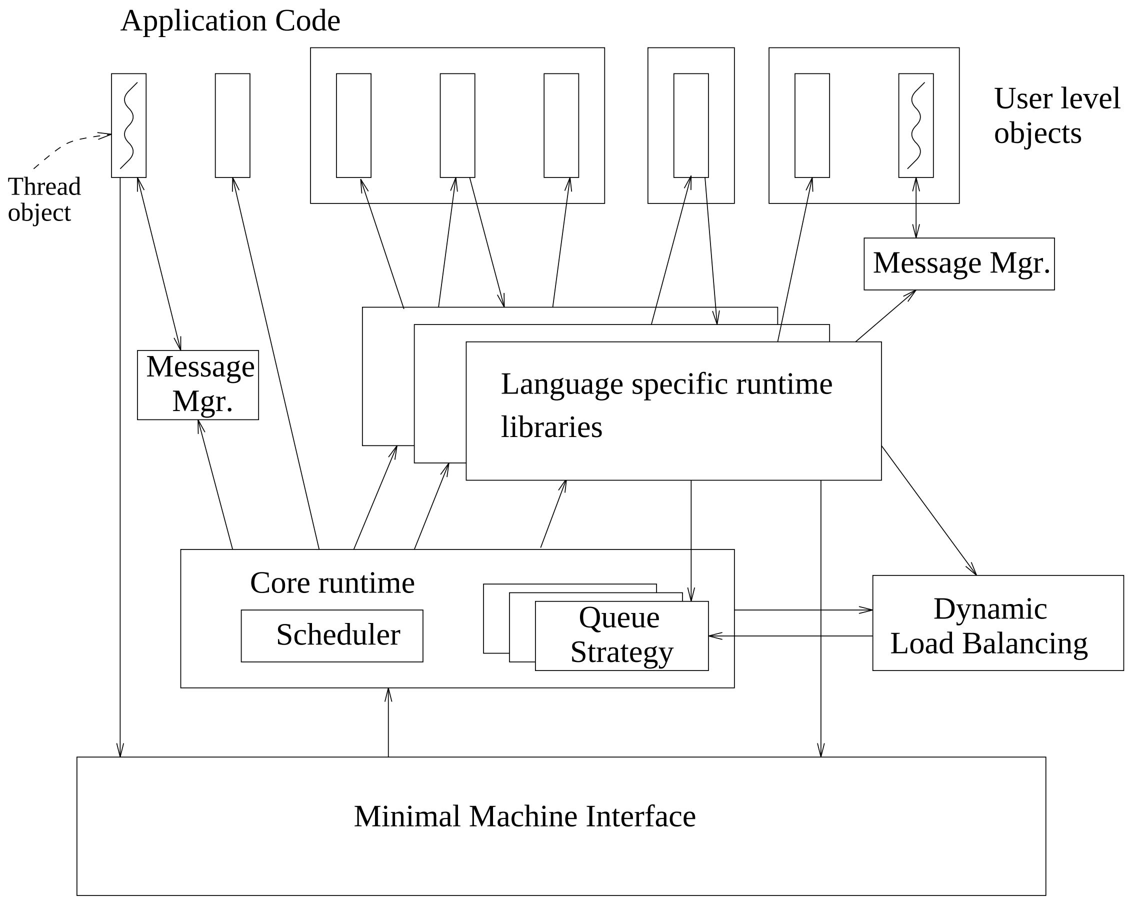
- Used by Concurrent Objects, Thread based and handler-based languages/libraries if and when they need it
- Generalized handler-based messages
- Scheduler maintains a single message queue
- Plug in different queue implementations e.g. fifo, prioritized

```
Scheduler()
{     while ( not done ) {
          msg = GetMsg() ;  // From network or queue
          (HandlerOf(msg))(msg) ; }
}
```

CONVERSE Architecture



CONVERSE Architecture



CONVERSE Machine Interface

Support messaging for SPMD, objects, threads

- Handler invoked on message delivery based on id in message : no tag matching/ordering overhead
- Synchronous send : message buffer reusable after send
- Asynchronous send : no waiting for message to leave processor, no copying
- Atomic terminal I/O, timers, etc.

CONVERSE Machine Interface

Extended Machine Interface : generic implementations, can be specialized for individual machines.

- Global pointers (Create / Get / Put)
- Interrupt (active) messages
- General Gather / Scatter
- Processor Groups
- Parallel I/O

Language Runtimes on Converse

- Paradigms supported
  - Message Driven Objects:
  - SPMD (PVM, NXLib)
  - Threads (with message passing)
- Common facilities needed for supporting paradigms/language runtimes
  - Message managers
  - Thread objects
  - Others, ...

Message Managers

- Hold messages that have arrived until asked for
- Essentially, indexed mailboxes
- Provide calls for tag based message
  - insertion, retrieval and probing
- Wildcarded retrieval is allowed

Thread Objects

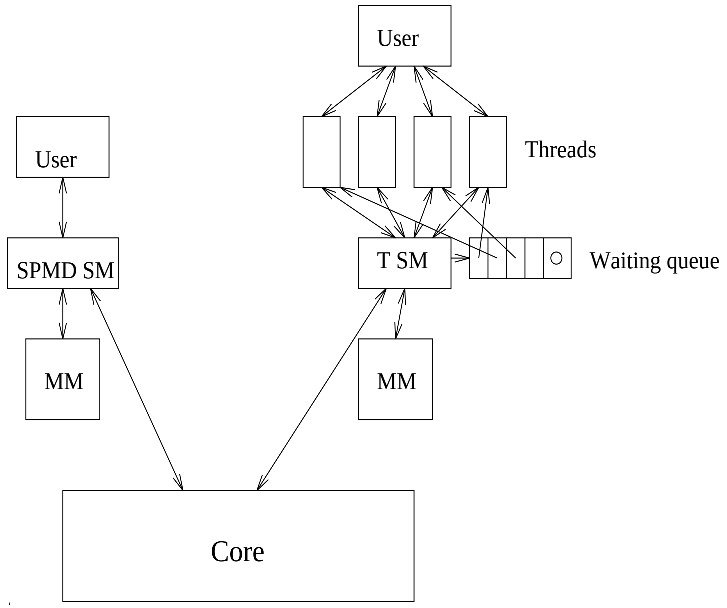
- Thread functionality seperated modularly
  - Thread state (stack, registers ..) $\rightarrow$ thread object
  - Scheduler (queue of threads)
  - Synchronization mechanisms (locks, condition variables ..)
- Thread objects provide the following functionality :
  - **create**, **yield**, **exit**, **suspend**, **resume** and **awaken**
- Portable implementation with low overheads

Thread Scheduling:

- Each thread can have a different scheduler
- Different types of schedulers can be built
  - round robin, hierarchical, message driven
- Threads can also be scheduled using Converse scheduler

PVM messaging, Threaded PVM

- Implementation of PVM messaging calls using Converse MI
- Message buffering, ordering layered over Converse Core



- Molecular dynamics program – "namd" – being ported to this layer.

Converse Status

- Languages implemented using Converse
  - PVM-messaging, NXLib, Simple Messaging
  - Threaded SM, Threaded PVM
  - Charm and Charm++
  - Parallel Import: a parallel discrete event simulation language
  - Several small experimental languages
  - Planned : (threaded) MPI, Multipol, ...

Converse Status

- Converse runs on
  - Workstation networks (using UDP/IP, ATM, FM/Myrinet)
    - Reliable UDP implementation using packetization and windowing.
    - FM : packetization for large messages.
  - IBM SP-2, Cray T3D/FM, Intel Paragon, Convex Exemplar, nCube, CM-5, ..

Related work

- Interoperability within same paradigm: HPF-MPI
- Nexus:
- PORTS: Portable Run-Time System (working group)
- Chant: message passing threads
- tPVM : threaded PVM
- Messaging layers: FM, Active messages, ...
- HPC++ : data parallel, threads, objects

CONVERSE Messaging performance

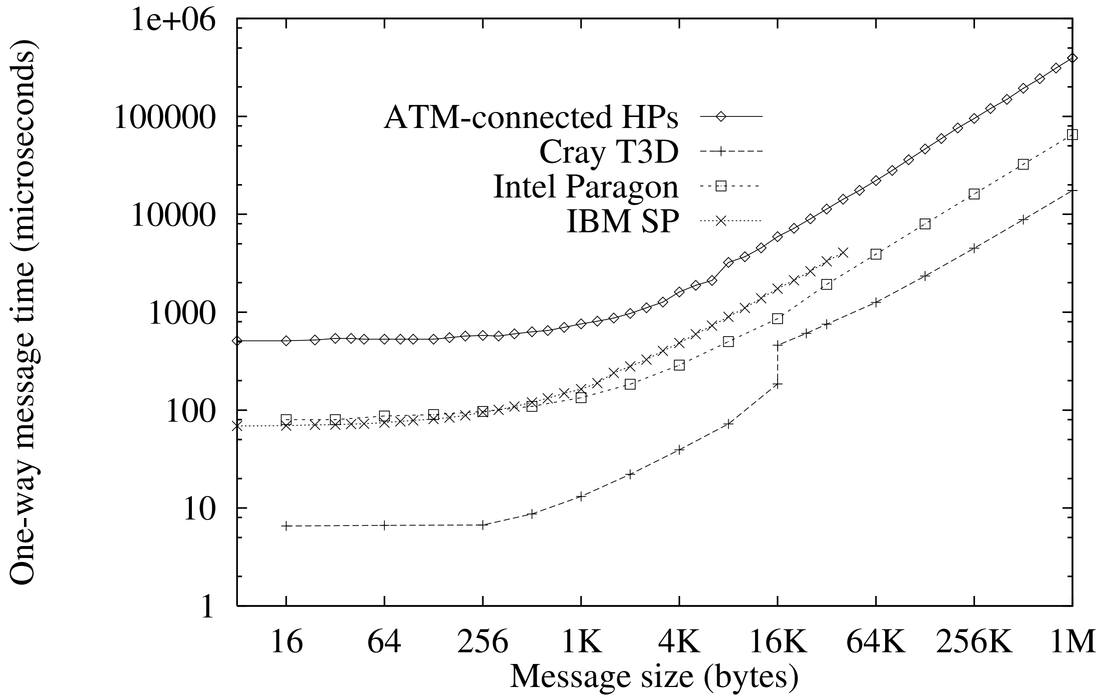


Figure 1: Performance on various parallel machines

CONVERSE Messaging/Scheduling performance

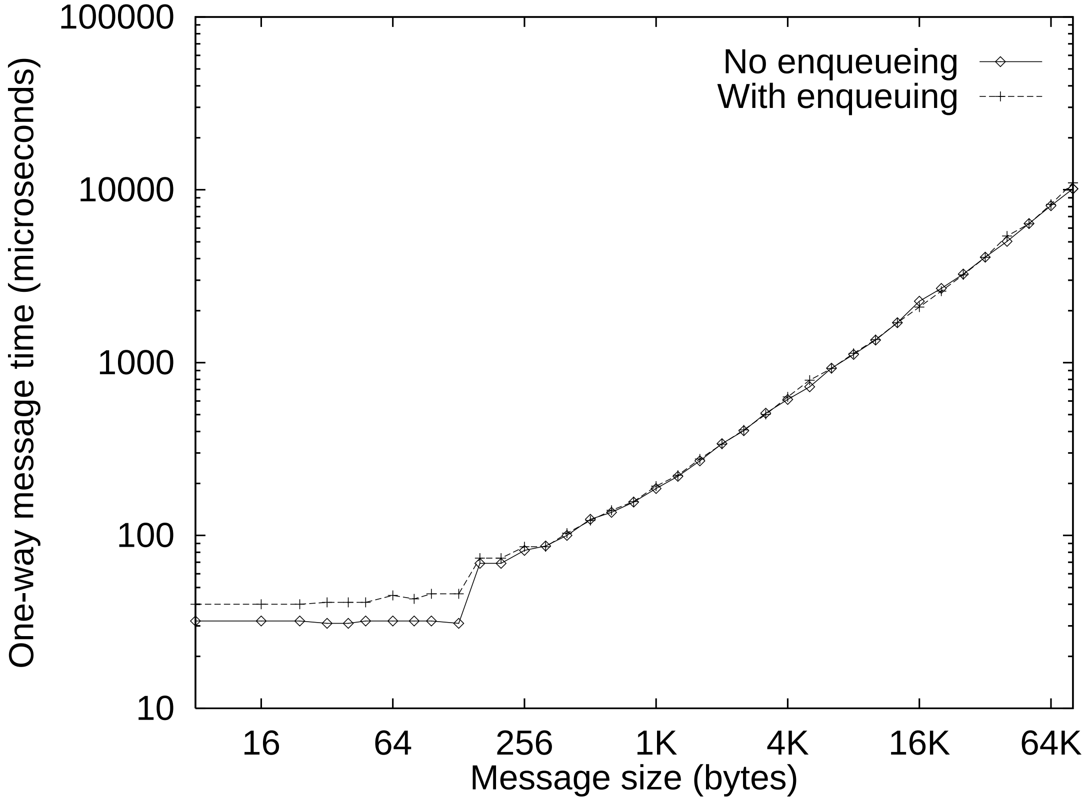


Figure 2: Performance on Sun workstation networks using FM Myrinet Switch

Summary

- Converse allows modules from different languages to be used in a single application.
- Minimum overheads (need based cost),
- Component based approach
- highly portable “thread component” implementation.
- Different languages have been ported on to Converse
  - More will be ported soon
- Currently runs on a number of machines
- For more information, take a look at our WWW site : http://charm.cs.uiuc.edu

Future Work

- Object-based languages: define "unversal" representation of object ids.
- Live as an ORB (CORBA compliance)
- Improve implementation (Vendor cooperation)
- Generate consensus among implementors/language designers
- MPI implemenations
- (Multitudes of) Language implementations, Collaborations
- Multi-paradigm, multilingual parallel programming?